MARTE/pCCSL: Modeling and Refining Stochastic Behaviors of CPSs with Probabilistic Logical Clocks

Dehui Du, Ping Huang, Kaiqiang Jiang, Frédéric Mallet, Mingrui Yang

MARTE/pCCSL: Modeling and Refining Stochastic Behaviors of CPSs with Probabilistic Logical Clocks

Dehui Du[1], Ping Huang[1], Kaiqiang Jiang[1], Frédéric Mallet[1,2,3], and Mingrui Yang[1]

[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[2] Univ. Nice Sophia Antipolis, I3S, UMR 7271 CNRS, France
[3] INRIA Sophia Antipolis Méditerranée, France

Abstract. Cyber-Physical Systems (CPSs) are networks of heterogeneous embedded systems immersed within a physical environment. Several ad-hoc frameworks and mathematical models have been studied to deal with challenging issues raised by CPSs. In this paper, we explore a more standard-based approach that relies on SysML/MARTE to capture different aspects of CPSs, including structure, behaviors, clock constraints, and non-functional properties. The novelty of our work lies in the use of logical clocks and MARTE/CCSL to drive and coordinate different models. Meanwhile, to capture stochastic behaviors of CPSs, we propose an extension of CCSL, called pCCSL, where logical clocks are adorned with stochastic properties. Possible variants are explored using Statistical Model Checking (SMC) via a transformation from the MARTE/pCCSL models into Stochastic Hybrid Automata. The whole process is illustrated through a case study of energy-aware building, in which the system is modeled by SysML/MARTE/pCCSL and different variants are explored through SMC to help expose the best alternative solutions.

Keywords: cyber-physical systems, MARTE, pCCSL, stochastic hybrid automata, energy-aware building, statistical model checking

1 Introduction

Cyber-Physical Systems (CPSs) combine digital computational systems with surrounding physical processes and can be viewed as a network of embedded systems where a (large) number of computational components are deployed within a physical environment [25]. Each component collects information about and offers services to its environment (e.g., environmental monitoring, healthcare monitoring and traffic control). This information is processed either within the component, in the network or at a remote location (e.g., a base station), or in any combination of these. The prominent characteristic of CPSs is that they have to meet a multitude of quantitative constraints, e.g., timing constraints, energy consumption, memory usage, communication bandwidth, QoS, and often with uncertain environment or user behaviors. So how to model and verify CPSs still remains a challenging problem. Now CPSs are spreading in numerous applications with a large societal impact, ranging from automotive systems, manufacturing, medical devices, automated highway systems (AHS) [17], air traffic control systems [37], personal and medical devices [22], Smart Grids and Smart Building, etc.

In the literature we find that both a variety of engineering modeling languages (lots of them are UML/SysML/MARTE-based) and a bunch of formal models (e.g., timed automata, hybrid automata, Petri nets, synchronous languages) provide a good support for formal verification. However, the integration of industry standards with verification frameworks is still in its infancy. For instance, a classical flow consists in adorning a UML state machine with some annotations and then transforming it into a timed automata for verification. We intend to go further by combining together several models of various kinds to cover heterogeneous aspects of the systems before transforming them into a language amenable to verification. While several frameworks inspired by Ptolemy [31] address the important issue of heterogeneity, most of them propose an ad-hoc environment and notation, while here we start from UML/SysML/MARTE models. The aim of our work is to facilitate the modeling CPSs with standard-based modeling language.

We consider that the UML, as a general-purpose language, provides a variety of models to cover lots of aspects of CPSs, structural aspects with structured classifiers and components, state-based models with state machines and data-flow models with activities. Because we target embedded systems, we use the MARTE profile, which appears as the best choice [8] for a UML-based solution. In particular we focus on its subprofiles covering time, allocation, non-functional properties (NFP)

(like power or energy consumption) and Performance Analysis Modeling (PAM). Because CPSs combine discrete and continuous aspects, we follow the lead of other works [32] and combine MARTE with SysML. In particular, we use SysML parametrics to capture the equations that link the energy to time and power.

As in any UML-based models, the relationships among models and the consistency is of paramount importance. We claim that logical clocks [23], just like tagged structures [24], provide a good abstraction to link different models together. Indeed, logical clocks can be used as activation conditions [3] of different models. Clock constraints then define a coordination model to constrain the joint execution of these models. The time subprofile of MARTE extends the UML with logical clocks that can then be used to control the different interactions between the models, e.g., the relationships between a state transition and a part in a structured classifier, or the start of an action and the sampling step to integrate the energy and compute the power consumption, but also relationships between UML and SysML models.

While MARTE extends UML with the notion of logical clock, its companion language CCSL [1] offers a syntax to build clock constraints. CCSL as a declarative language helps build a specification that can be refined when new constraints from the environment, or the platform or the application become available. There may be several implementations that satisfy a CCSL specification, a classical approach [15] consists in defining a policy (e.g., non-deterministic choice or as soon as possible) to decide which solution to retain. Another solution, which is explored here, consists in extending CCSL with stochastic constructs and probability. Such constructs help pick one solution instead of another one by giving the likelihood that a clock ticks.

Once the SysML/MARTE model is built, we propose to analyze the resulting model through a transformation to Stochastic Hybrid Automata and to use Statistical Model Checking (SMC) [6, 27, 33, 39] to explore different solutions. To illustrate our approach, we take the example of a energy-aware building and show how to explore and compare alternative solutions.

Our contributions are 1) to propose an extension of CCSL, called pCCSL, with stochastic constructs and probability to drive the exploration of alternative solutions when building clock specifications; 2) To show how pCCSL specifications can augment UML/SysML/MARTE models to link together several models; 3) To propose a structural transformation into SHA according to the semantics of pCCSL so as to perform evaluation on the alternative solutions with statistical model checking, which helps designers refine system models. The whole process is illustrated on the example of an energy-aware building.

The remainder of this paper is organized as follows. Section 2 introduces CCSL and our proposed extension, pCCSL. Section 3 introduces our case study and makes a joint use of UML, SysML and MARTE to capture different aspects of this model. Section 4 proposes some transformation rules to transform MARTE/pCCSL into SHA. Finally we position our work with respect to related works before concluding and discussing possible future extensions.

2 pCCSL: the Probabilistic Extensions of CCSL

2.1 Syntax of pCCSL

We first recall the basic constructs of CCSL [1] and then further describe the proposed extensions (see Fig. 1).

Core constructs of CCSL. A specification is made of clock relations and declarations. Relations prevent some clocks from ticking depending on configurations. Declarations are meant to declare new clocks, either to capture events of the system or to build intermediate clocks based on other ones. The two basic CCSL relations are *subclocking* and *causality*. The former one is inspired by synchronous languages and prevents one clock (the subclock) from ticking when its superclock cannot tick. The superclock is said to be coarser and the subclock is finer than the other one. The latter one is akin to the causality relation in event structures. It prevents a clock (the effect) from ticking when another clock (its cause) has not ticked. The cause is said to be faster than the effect (which is then slower). This typically represents first-in-first-out (FIFO) constructs, e.g., one cannot read a data before it has been written. The other relations (*precedence*, *synchrony*, *exclusion*) are derived.

When declaring a clock, one can specify (if required) how it is related to other clocks. This is done using *clock definitions*, e.g., *filters*, or *expressions*.

Filters allow for precisely defining a subclock based on its superclock. For instance, a *Periodic-Filter* makes one clock tick every p ticks of its superclock. The *Select* expression selects an interval (possibly not closed) of ticks to the superclock at which the subclock ticks.

Function-expression builds a new clock based on two (or more clocks). *Union* builds the coarsest clock that is a super clock of two other clocks, whereas *Intersection* builds the finest subclock of two clocks. *Inf* (resp. *Sup*) builds the slowest (resp. fastest) clock that is faster (resp. slower) than two clocks.

The *Sampling* takes a triggering and sampling clock and builds the fastest clock slower than the triggering clock and subclock of a sampling clock.

One can also declare a period and optionally a jitter for clock making reference to an ideal physical clock. This information related to physical time is not used in CCSL clock calculus but is a mere annotation to display results with a scale that is meaningful to the user.

⟨*specification*⟩ ::= { (⟨*relation*⟩)* (⟨*declarations*⟩)*}

⟨*relation*⟩ ::= c ⊆ c ['rate:' ⟨*nat*⟩ '/' ⟨*nat*⟩] (subclocking)
| c '<=' c ['size:' ⟨*nat*⟩] (causality)
| c '<' c ['size:' ⟨*nat*⟩] (precedence)
| c '=' c (synchrony)
| c '#' c (exclusion)

⟨*declaration*⟩ ::= 'Clock' c [⟨*definition*⟩]

⟨*definition*⟩ ::= 'is' ⟨*expression*⟩
| ⟨*filter*⟩
| 'period' ⟨*real*⟩ ['jitter' ⟨*real*⟩]
| 'with probability' ⟨*prob*⟩
| 'with distribution' ⟨*real*⟩ [c]

⟨*expression*⟩ ::= '0' | '1'
| c ⟨*operator*⟩ c

⟨*filter*⟩ ::= c
| 'every' ⟨*nat*⟩ c
| ':=' c ['from' ⟨*nat*⟩] ['to' ⟨*nat*⟩]

⟨*operator*⟩ ::= 'inf' | 'sup'
| 'inter' | 'union' | 'minus'
| 'sampledBy'

⟨*nat*⟩ ::= (* natural number constant *)

⟨*prob*⟩ ::= (* real number between 0 and 1 *)

Fig. 1: Grammar of pCCSL

Proposed extensions for pCCSL. The grammar of pCCSL is shown in Fig. 1. There are three proposed extensions marked with red fonts. The first consists in adding a *rate* parameter to the subclock relation. It specifies the rate at which the subclock ticks compared to its superclock. It corresponds to a probability for the subclock to tick when its superclock ticks. This is an intermediate solution between saying nothing, which may imply that the subclock never ticks, and completely deciding when the subclock ticks, with a filtering expression for instance.

The second proposed extension consists in assigning a *probability* to a clock, to replace an expression. Rather than giving a deterministic expression that says when the clock may or cannot tick, we give the probability that the clock ticks. As we show later, the consistent integration of this probabilistic clock with the semantics of the other relations is not trivial and requires some particular caution. However, this is useful to help CCSL clock calculus picking one solution instead of another one when several solutions are possible.

The third and last extension consists in giving a parameter λ to control when a clock ticks or not. The lambda parameter makes a clock tick according to an exponential distribution. The probability that the clock ticks at the time less than x is $1 - e^{-\lambda x}$. x here is either relative to the

ticks of another clock or, if no reference clock is provided, relative to an absolute ideal physical-time clock. In the latter case, the information is a mere annotation only used to interpret the ticking of the clock according to an ideal physical time reference but is not used in the clock calculus process.

Note that these three extensions cannot add new solutions, they are just meant to reduce the set of solutions in case of several possible solutions. The probability and distribution constructs help us decide on the likelihood for a clock to tick (when allowed by the other constraints).

2.2 Semantics of pCCSL

A constraint specification $spec = \langle C, Cons \rangle$, is a tuple where C is a set of clock names and $Cons$ is a set of constraints. The semantics of each individual constraint is given by a special form of transition system called clock-labeled transition systems.

Definition 1 (Labeled Transition System). *A* labeled transition system *(lts) is a structure* $\langle S, A, \longrightarrow \rangle$ *consisting of a set* S *(of elements,* s*, called states), a set* A *of labels, and a transition relation* $\longrightarrow \subseteq S \times A \times S$*.* $s \xrightarrow{a} s'$ *is used to denote* $(s, a, s') \in \longrightarrow$*.*

A clock-labeled transition system is a lts where each label is a set of clocks. From each state, there are maximum 2^n outgoing transitions, where $n = |C|$ is the number of clocks. Each transition corresponds to a particular *configuration* of ticking clocks. Transition systems may have an infinite number of states.

Definition 2 (Clock-Labeled Transition System). *A* clock-labeled transition system *(clts) is a structure* $\langle S, C, \longrightarrow \rangle$ *where* $\langle S, 2^C, \longrightarrow \rangle$ *is a labelled transition system.*

To capture the semantics of the proposed extension we extend the clts by adding a probability to each transition.

Definition 3 (Probabilistic CLTS). *A* probabilistic clock-labeled transition system *(pclts) is a clts with an extended transition relation* $\longrightarrow \subseteq S \times 2^C \times P \times S$*, where* $P \subseteq \mathbb{Q}$ *is a real number between 0 and 1 (i.e., a probability).*

For a given transition $t = (s, \Gamma, p, s') \in \longrightarrow$, $\pi(t) = p$ denotes the probability p that the transition t is fired.

For a pclts $\langle S, C, \longrightarrow \rangle$, we call $s^\bullet$ the set of all transitions whose source is s:

$$s^\bullet = \{(s, \Gamma, p, s') \in \longrightarrow\}$$

Note that $s^\bullet$ can never be empty since it is always possible to do nothing in CCSL, i.e., $(s, \emptyset, p, s)$ is always in $\longrightarrow$ for all $s \in S$ and for some value p.

Given a clock $c \in C$, let us call $s^\bullet_c$ the set of all transitions whose source is s and such that the clock c ticks:

$$s^\bullet_c = \{(s, \Gamma, p, s') \in \longrightarrow |\ c \in \Gamma\}$$

For a pclts to be well-formed, it must satisfy the two following conditions:

$$\forall s \in S, \sum_{t \in s^\bullet} \pi(t) = 1 \tag{1}$$

$$\forall s \in S, \forall c \in C, \sum_{t \in s^\bullet_c} \pi(t) = p_c \tag{2}$$

In Eq. 2, for each clock $c \in C$, the probability p_c is either manually assigned by the user with a declaration 'Clock c probability p', or derived using the rate in a subclocking relation or assigned to the default value $1/|s^\bullet|$ otherwise.

A 'normal' clts can be seen as a probabilistic clts where all the probabilities are assigned with default values $1/|s^\bullet|$ for all the states $s \in S$.

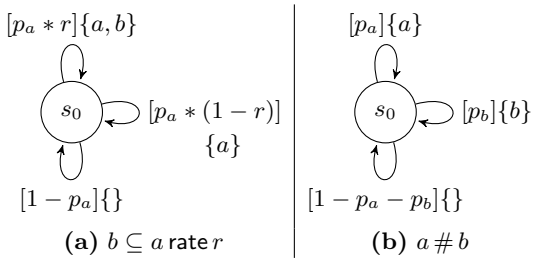
Subclocking and synchrony. Let $a, b \in C$ be two clocks and $r \in \mathbb{Q}$ a real number such that $0 \leq r \leq 1$. The *subclocking* relation (see Fig. 2(a)), $b \subseteq a \,\mathsf{rate}\, r$ is defined as a pclts $\langle \{s0\}, \{a, b\}, \longrightarrow_\subseteq \rangle$, such that $\longrightarrow_\subseteq = \{(s0, \{\}, 1 - p_a, s0), (s0, \{a, b\}, p_a * r, s0), (s0, \{a\}, p_a * (1 - r), s0)\}$, where $p_a \in \mathbb{Q}$ is the probability assigned to clock a. Let us note that Eq. 1 is satisfied since $\sum_{t \in s0^\bullet} \pi(t) =$

$(1-p_a)+(p_a*r)+(p_a*(1-r))=1$. Eq. 2 is also satisfied since $\sum_{t\in s0^{\bullet}_{b}}\pi(t)=p_a*r=p_b$ and $\sum_{t\in s0^{\bullet}_{a}}\pi(t)=(p_a*r)+(p_a*(1-r))=p_a$.

If no probability was assigned then the default is 2/3. If no rate is assigned, then r defaults to 1/2. With default values, each one of the three transitions has a probability of 1/3, i.e., each transition has the same probability to be fired. The transition $\{b\}$ however has a probability of 0 since it would contradict the subclocking relation.

Note that if both the probability of a is given and the rate of b relative to a is given, then $p_b=p_a*r$. In any other cases, the specification is ill-formed.

The synchrony constraint is a special case of subclock defined as follows $a=b\equiv b\subseteq a$ rate 1, which implies $p_a=p_b$.



(a) $b\subseteq a$ rate r **(b)** $a\,\#\,b$

Fig. 2: pclts for subclocking and exclusion

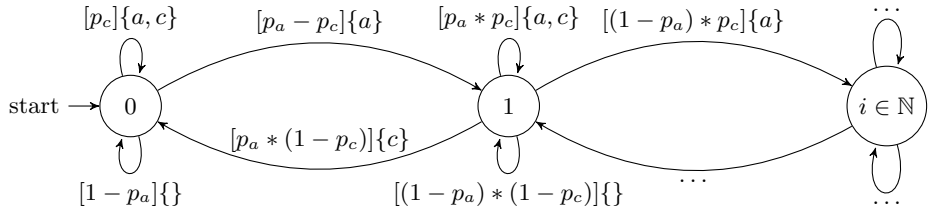


Fig. 3: pCCSL causality (infinite state pclts): $a\leq c$.

The exclusion constraint (see Fig. 2(b)), $a\,\#\,b$ is defined similarly using the following transition relation $\longrightarrow_{\#}=\{(s_0,\{\},1-p_b-p_a,s_0),(s_0,\{b\},p_b,s_0),(s_0,\{a\},p_a,s_0)\}$. Again the two consistency rules are satisfied.

Causality. Let $a,c\in C$ be two clocks. The *causality* relation (see Fig. 3), $a\leq c$ is defined as a pclts $\langle\mathbb{N},\{a,c\},\longrightarrow_{\leq}\rangle$, such that $\longrightarrow_{\leq}=\bigcup_{i\in\mathbb{N}}s_i$, where $\mathbb{N}$ is the set of natural numbers, $s_0=\{(0,\{\},1-p_a,0),(0,\{a,c\},p_c,0),(0,\{a\},p_a-p_c,0)\}$, and for all $i\in\mathbb{N},i>0$, $s_i=\{(i,\{\},(1-p_a)*(1-p_c),i),(i,\{a,c\},p_a*p_c,i),(i,\{a\},p_a*(1-p_c),i+1),(i,\{c\},(1-p_a)*p_c,i-1)\}$. $p_a,p_c\in\mathbb{Q}$ are the probabilities assigned to clocks a and c respectively.

Note that whatever the values of p_a and p_c, then a will occur more frequently than c since c cannot occur alone in state 0 and to reach state 0, a and c must have occurred exactly the same number of times.

When a size is associated with the causality constraint, then the transition system becomes finite and is such that the number of states equals to $size+1$. With size 1, there are two states, with size 2 there are three states and so on.

The *precedence* is very similar to *causality* except that in state 0 the simultaneous occurrence of a and c is forbidden. The whole semantics of the operators is available in [30] and we just give here the ones used in this paper and that have been extended with stochastic constructs.

2.3 Composition

Each constraint is expressed as a clts and a specification is then captured as the synchronized product [2] of all the clts. The process for standard CCSL constraints is explained in detail in [30].

With the proposed extensions, for each clts $L = \langle S, C, \longrightarrow_L \rangle$ that results from the synchronized product and such that C is the union of all the clocks of all the composed clts, we derive a pclts $P = \langle S, C, \longrightarrow_P \rangle$ such that the two consistency rules (Eq.1-2) are satisfied.

For instance, let us consider the following pCCSL specification:

1. Clock a with probability p_a
2. Clock $b \subseteq a$ rate: r
3. Clock $a \leq c$ size: 1

The semantics of each constraint is captured with a clts which are then composed through a synchronized product. Then the probabilities are added to the transitions resulting in the pclts shown in Fig. 4.

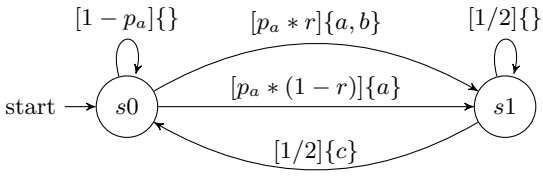


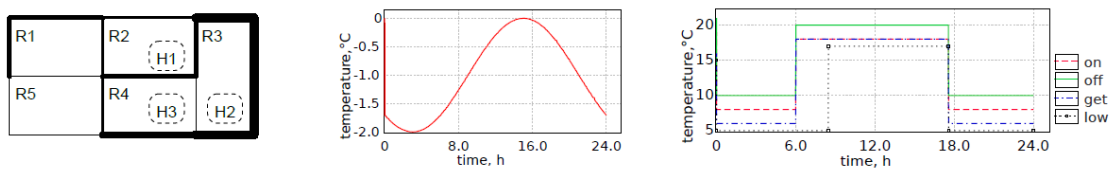
Fig. 4: Example of composition through synchronized product.

Note that Eq. 1 is satisfied since $\sum_{t \in s0^\bullet} \pi(t) = (1-p_a) + (p_a * (1-r)) + (p_a * r) = 1$ and $\sum_{t \in s1^\bullet} \pi(t) = 1/2 + 1/2 = 1$. Eq. 2 is also satisfied since $\sum_{t \in s0^\bullet_a} \pi(t) = (p_a * (1-r)) + (p_a * r) = p_a$ and $\sum_{t \in s0^\bullet_b} \pi(t) = p_a * r = p_b$. Since the probability of c is not given and c is the only clock that can tick from state $s1$ then the outgoing transitions are assigned with the default value $1/2$ since $|s1^\bullet| = 2$.

3 Modeling and Refining Energy-Aware Building with MARTE/pCCSL

This section describes the application of MARTE/pCCSL to a hybrid system benchmark, which addresses the control problem of the temperature of rooms in a given building with limited heaters. With MARTE/pCCSL, we can flexibly model every part of the system in multi-views (structure, equation, behavior, and clock constraints) with quite loose coupling between them. MARTE facilitates the modeling of Time and Non Functional Properties (NFP, here energy and temperature), and pCCSL provides precise and probabilistic time control through definition and constraints of logical clocks used between abstract specification and its refinements to help us expose the best alternative solutions. Next we briefly introduce the case of energy-aware building and then present its SysML/MARTE/pCCSL models in multi-views.

3.1 Energy-Aware Building Setup



(a) Rooms R_i with heaters H_k. (b) Outside temperature. (c) Dynamic user profile.

Fig. 5: Representation of building layout, outside temperature and dynamic user profile

As a starting point, consider a setup proposed by [11] as an extension of the challenging benchmark for hybrid systems model-checkers addressed in [20]. The benchmark consists of a building

layout with temperature dynamics, autonomous heaters and a central controller deciding which room gets a heater. The room temperature dynamics is described by a differential equation:

$$T_i' = \sum_{j \neq i} a_{i,j}(T_j - T_i) + b_i(u - T_i) + c_i h_i$$

where T_i and T_j are the temperatures in rooms i and j respectively, u is the environment temperature and h_i is equal to 1 when the heater is turned on in the room i and 0 otherwise. The building layout is encoded by an adjacency matrix a where $a_{i,j}$ is a heat exchange coefficient between rooms i and j. The heat exchange with an environment is encoded in a separate vector b, where b_i is the energy loss coefficient for room i. An energy supply from a heater is encoded as a vector c, where c_i is a power coefficient for room i. Fig. 5(a) shows a benchmark building configuration instance (`HEAT15` in [20]) with rooms and heaters, where the wall thickness corresponds to an isolation defined by a and b. The definition of matrix a, vectors b and c can be found in [11].

Each heater is equipped with a bang-bang controller configured to turn on the heating ($h_i := 1$) when the temperature T_i is below threshold on_i and turn off ($h_i := 0$) when the temperature T_i is greater than off_i. Whenever the heating is turned on, the heaters consume an amount of power denoted by vector pow. The central controller can switch-over the heating from one room to another. The room is said to be needing a heater if the temperature drops below its get threshold and it is said to be outside comfort zone if the temperature drops below low. To decide when the heating can be switched over, we consider a control strategy, which is based on heuristics that the temperature difference between rooms should not be too high. To be precise, if $room_i$ whose temperature $T_i \leq get_i$ has no heater while $room_j$ has a heater, the heater can move from $room_j$ to $room_i$ when the difference $T_j - T_i \geq dif_i$. To reduce the non-determinism further, we consider probabilistic choices between the possible room destinations denoted by probabilistic weights imp. The temperature thresholds for each room used above also refer to [11].

Further we propose to augment this setup with specific weather conditions and different user profiles to make a more realistic case for optimizing the energy consumption. The benchmark [20] assumes that the environment temperature is within a range between $0°C$ and $-2°C$ without any specific dynamics. Fig. 5(b) shows one daily cycle and Fig. 5(c) shows the user profile with dynamic temperature thresholds in a day. To this user profile, we have two different refinement versions, one of which is a more complex user profile with probabilistic choice that is explained later to show the usage of MARTE/pCCSL and probabilistic clock and to demonstrate how to refine the models.

3.2 MARTE/pCCSL Models of Energy-Aware Building

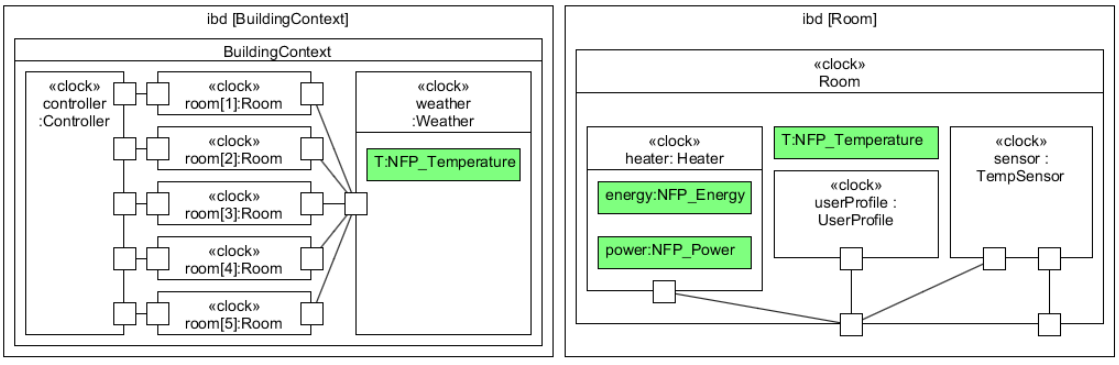


Fig. 6: Structure of an energy-aware building (Internal Block Diagram)

Structure Modeling As described above, the structure of our energy-aware building is modeled with SysML Internal Block Diagrams shown in Fig. 6. *BuildingContext* represents the whole context that consists of five rooms and a central controller as well as the environment such as weather. *Room* represents the common template of five rooms, each of which contains a *heater*, a *TempSensor*,

and a *UserProfile* for user-defined temperature conditioning thresholds. We limit the maximum number of heaters turned on at a time to three, though each room has a heater. Each part in the energy-aware building is stereotyped by *clock* that means their behaviors should be controlled by certain clock constraints (specified in pCCSL). For instance, *controller* monitors the information of *rooms* with the constraint $monitor = sensor_i$ (i is from 1 to 5), representing that monitor clock in *controller* and sensor clock of each *room* is synchronous. Besides such discrete control behavior, continuous behavior such as temperature change of each *room* as well as outside environment (*weather*) is constrained by clock, that is, continuous behavior is discretized by clocks. We use several *nfpTypes* to define related variables like *Weather.T* and *Room.T* as *NFP_Temperature*, *Heater.energy* as *NFP_Energy*, *Heater.power* as *NFP_Power* so that NFP properties like *unit* can be used. Several *nfpTypes* are imported from MARTE Library (*NFP_Energy* and *NFP_Power*), and *NFP_Temperature* is defined by us using MARTE constructs (units, dimension and nfpType stereotypes). The definition is simple and similar to most *nfpTypes* defined in MARTE Library, so we do not discuss the details.

Time/Clock Domain Fig. 7 shows the time domain of the whole system. Seven clocks are defined to constrain the time of the system and three of them are defined as probabilistic ones with the keyword **rate**. We define a new *clockType* called *BuildingClock* which serves as a base time of the whole system. *BuildingClock* is discrete and owns a read-only attribute *resolution*. *sysClk*, *hour*, *stepClk* are three instances of *BuildingClock* with their own time units and resolutions for different usage. *sysClk* with high resolution (0.01s) is used in precise control process; *hour* with low resolution (1h = 3600s) is used to specify user profile. *stepClk* is similar to *sysClk*, but is mainly used for discretizing continuous behavior. Then we import *idealClk* from MARTE TimeLibrary to constrain our clock instances with pCCSL. Besides, we define three **probabilistic subclocks** *substepClk*, *refinehour1* and *refinehour2*. *substepClk* is the subclock of *stepClk* with rate 2/3. *refinehour1* and *refinehour2* is the subclock of *hour*, where *refinehour1* is used to describe user behaviors when the user may need to go out for meeting in the afternoon as well as *refinehour2* is used when the user will not go out and just work.

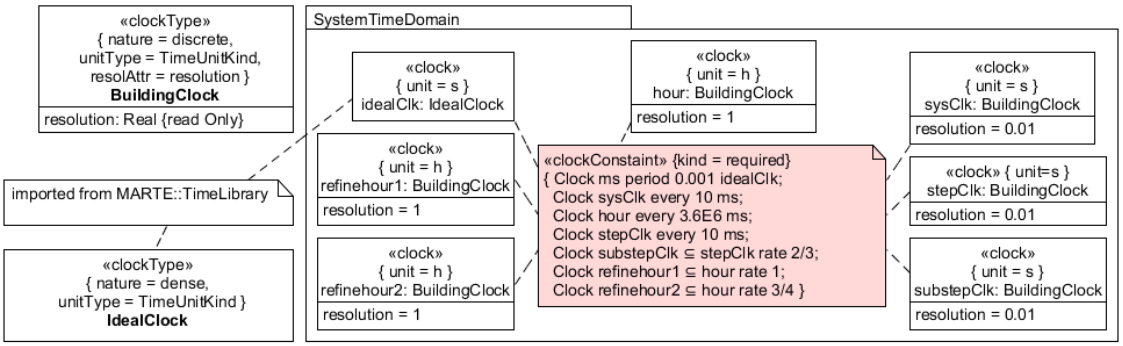


Fig. 7: Time domain of an energy-aware building

Behavior Modeling The system behaviors specified with pCCSL specifications are shown in Fig. 8. We present these activities stereotyped by *timedProcessing*, of which the first represents monitoring and scheduling behaviors of the system and the rest represent pre-defined user behaviors (user profiles). System behavior references *sysClk* and the whole control process is triggered by a *timedEvent* called *monitorEvent* that occurs every 120 seconds. Each time the system *readSensorData* (e.g. temperature of rooms) and also *readProfileData* (e.g. whether user stays in the room or not). Then the system starts *Schedule* process made of three sequential actions: *getNeedList* (to collect need and heating information), *chooseMostImportant* (to find the room that needs to be heated most according to the strategy if more than one room is in the list), and *changeHeatingConfig* (that changes heating parameter of rooms or mode of heaters). Each action is also stereotyped by *timedProcessing* and may include complex operations implemented in code. The *chooseMostImportant* contains the code implementation of scheduling strategy described in the last subsection.

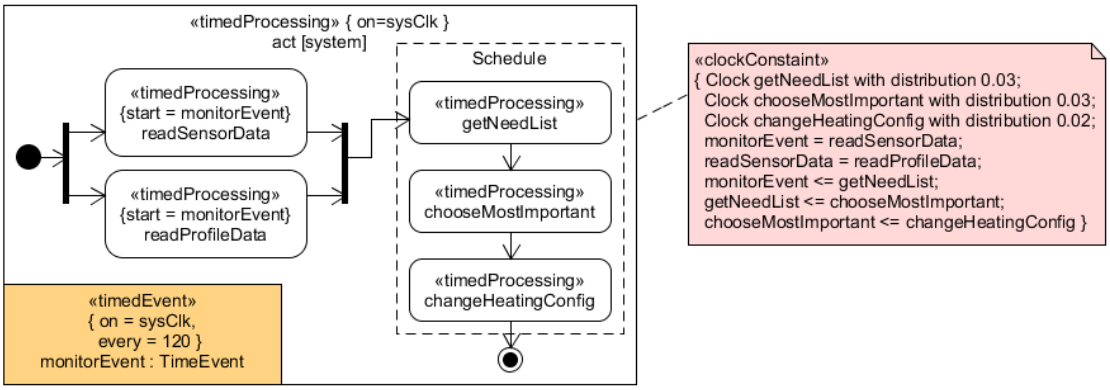


Fig. 8: Activity diagram for system behaviors

The *changeHeatingConfig* contains the code that will set necessary parameters such as *Heater.on* (for denoting the running mode of heater), *HeatingVector* (for denoting the heated state of room), and *Room.cold* (for denoting the discomfort state of user) and so on. The pink part in Fig. 8 is the pCCSL specification including three kinds of constraints:

- distribution clock (e.g. *Clock getNeedList with distribution 0.03* that means *getNeedList* occurs with an exponential time delay whose parameter $\lambda = 0.03$ on this clock) to describe the possibility of the unstable performance of sensors;
- clock synchrony (e.g. *monitorEvent* = *readSensorData* that means *monitorEvent* coincides with *readSensorData*);
- clock causality (e.g. *getNeedList* $\leq$ *chooseMostImportant* that means *getNeedList* is always followed by *chooseMostImportant*).

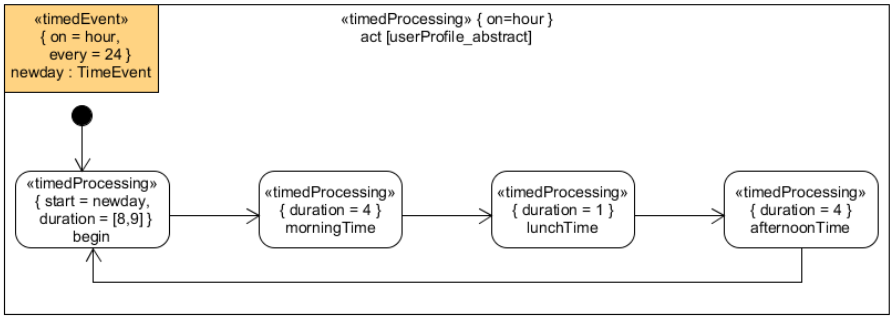


Fig. 9: Abstract activity diagram for user behaviors

Fig. 9 shows us the abstract user behaviors. *userProfile_abstract* references the clock *hour* to describe the abstract possible actions of user in a day. The *newday* occurs every 24 hours, representing the start of one day. User may arrive at the building between 8:00 and 9:00. The morning time lasts 4 hours and the afternoon time lasts 4 hours. In the middle, we have 1 hour to have lunch. Having this abstract specification, we have two kinds of refined models with subclocks of *hour*.

Refinement of Behavior Models The refinement processes are shown in Fig. 10 and Fig. 11. The refined version shown as Fig. 10 uses subclock *refinehour1* with rate 1. User arrives at the building between 8:00 and 9:00. After 4 hours, user may go for lunch between 12:00 and 13:00. After lunch, user can either go on working in the building (with a probability of 0.6) or go to a meeting (with a probability 0.4). If working in the afternoon, user may leave the building after 4 hours (i.e. between 17:00 and 18:00). If not, user will not go back to the building until next day.

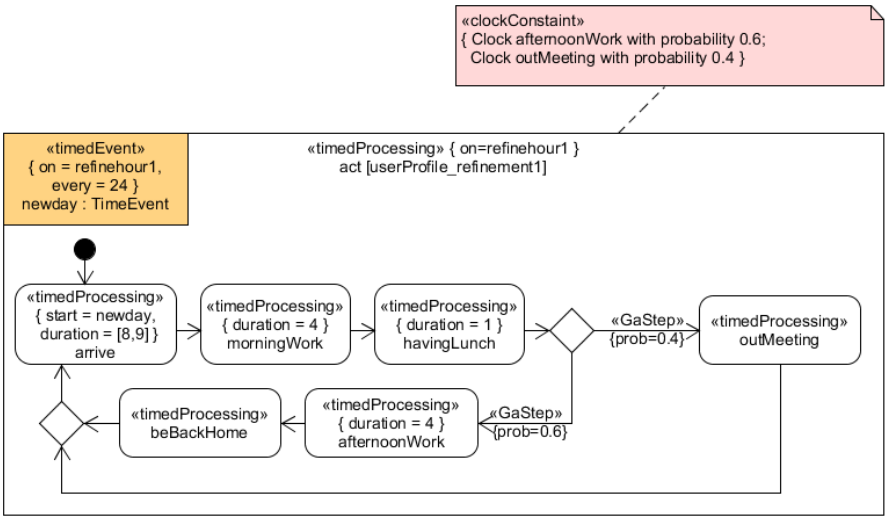


Fig. 10: Refined version 1 of activity diagram for user behaviors (refinehour1⊆hour, rate=1)

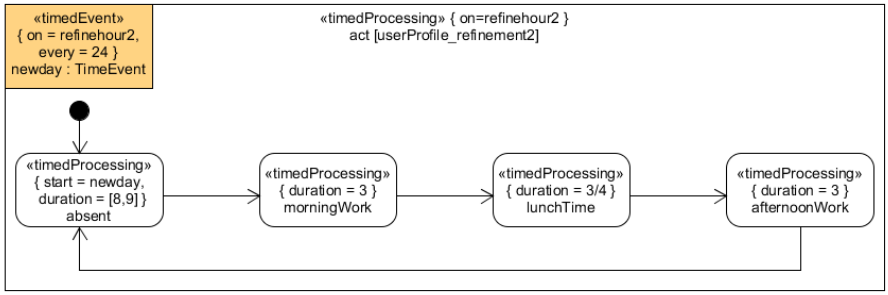


Fig. 11: Refined version 2 of activity diagram for user behaviors (refinehour2⊆hour, rate=3/4)

When user stays in the building, temperature thresholds are relatively higher than that when user does not, so that energy can be saved due to low temperature duration. As shown in the pink part of Fig. 10, pCCSL specifies the probabilistic clocks (e.g. *Clock outMeeting with probability 0.4* that means *outMeeting* may occur with a probability of 0.4). We also use MARTE stereotype *GaStep* from the Performance Analysis Modeling (PAM) subprofile to report that information on the UML model. The refined version shown as Fig. 11 uses subclock *refinehour2* with rate 3/4. The user doesn't need to work 8 hours a day, he/she only needs to work determined time given by subclock in the morning and afternoon, and the user can leave in the rest time and we neglect it. User may also arrive at the building between 8:00 and 9:00, and then, in 4 hours of morning time, according to the rate, user may work 3 hour randomly and the other time is rest time (neglected). In the afternoon, user may work 3 hour randomly, too. Thus, as we can see, the abstract specification and its refinements are connected logically by clock *hour* and its subclock *refinehour1* and *refinehour2*. Why we use user profiles to model is that user profiles are stochastic behaviors and we can refine these behaviors. The relation between Fig. 9, Fig. 10 and Fig. 11 is that Fig. 9 is the abstract specification of user profiles, while Fig. 10 is refinement1 and Fig. 11 is the refinement2 of the abstract specification. Lay down an abstract specification of some behaviors first, and then we can refine it with probabilistic clocks to model stochastic behaviors and various solutions.

Equation Modeling Continuous behaviors described by differential equations also play an important role. Most continuous variables need to be monitored in effective real-time control, such as **temperature** and **energy consumption** in this case. To model continuous behaviors of CPSs, the parametric models are used to present the Ordinary Differential Equations (ODE). As shown in Fig. 12, we consider four equations: *RoomTemperatureEquation* (that determines the temperature change of each room according to the adjacent room, weather, and heater), *HeaterEnergyEquation* (for monitoring energy consumption of each heater), *DiscomfortMonitorEquation* (for monitoring

discomfort value of users), *WeatherEquation* (that describes outside temperature curve). The former three equations use derivatives like *d(T)/d(t)* where *t* refers to the clock *substepClk* defined already as discretization step.

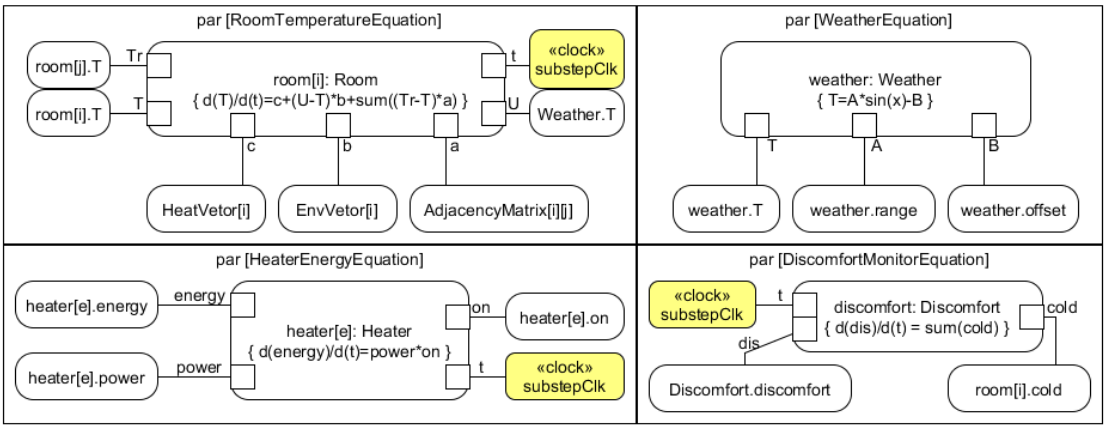


Fig. 12: Equations of continuous behavior of energy aware building (modeled as a SysML parametrics)

From this case study, we can find the reason why we use pCCSL rather than CCSL. Since pCCSL has the concept of rate, it can present the stochastic behaviors of the system effectivly and simply. CCSL does has logical time and ticks, but its tick position is certain. For instance, if A is a subclock of B and A ticks every four B, then the tick positions of A must be the fist position of every four consecutive B, that is the first, the fifth, the ninth and so on. But for pCCSL, if A is a subclock of B with rate 1/4, the tick position of A is stochastic. We can only ensure that A will tick only once in four consecutive B, but the certain position is unknown. Thus, it can be seen that pCCSL gives a global rate rather than a precise position as before, which can be used to model stochastic behaviors of CPS.

4 Transformation from MARTE/pCCSL to SHA for Evaluation

This section discusses the transformation approach from MARTE/pCCSL specification to Stochastic Hybrid Automata (SHA) used in Uppaal-smc [10]. The transformation mainly targets stochastic behaviors and continuous behaviors (ODE) in terms of SHA. The core elements of MARTE/pCCSL can be directly mapped to the elements of SHA. At last, we compare the energy consumption of users using Uppaal-smc aiming to help designers refine the design models.

4.1 Stochastic Hybrid Automata

Definition 4 (Stochastic Hybrid Automata). *A Stochastic Hybrid Automaton (SHA) H is a tuple* $H = (L, l_0, X, \Sigma, E, F, I)$*, where*

- L *is a finite set of locations,*
- $l_0 \in L$ *is the initial location,*
- X *is a finite set of continuous variables,*
- Σ *is a finite set of actions, and* $\Sigma = \Sigma_i + \Sigma_o$*, where* Σ_i *is the set of input actions,* Σ_o *is the set of output actions,*
- E *is a finite set of transitions, for each transition denoted by* $(l, g, \alpha, \varphi, l^{'}), l, l^{'} \in L$*,* g *is a predicate defined by* R^X *and action label* $\alpha \in \Sigma$*,* φ *is a binary relationship on* R^X*.*
- $F(l)$ *is a time delay function for each location* $l \in L$*,*
- I *is a finite set of invariants.*

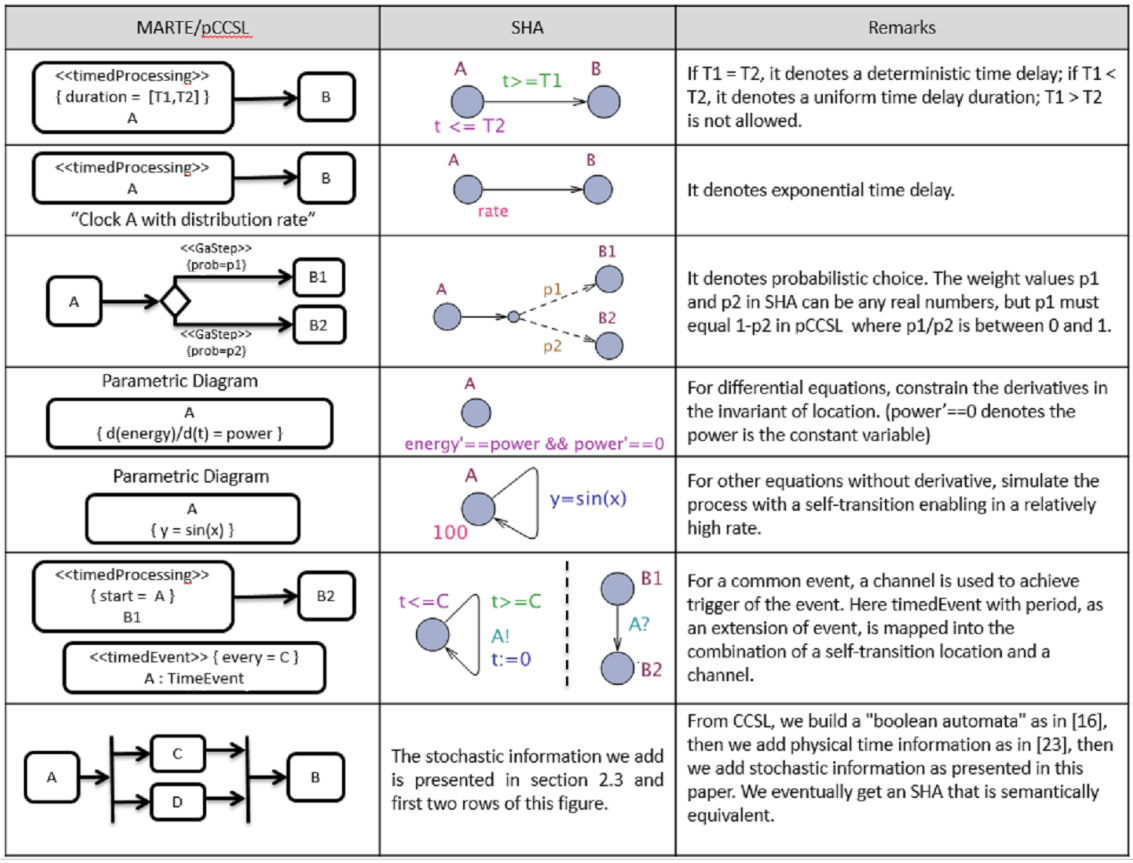


Fig. 13: Mapping rule from MARTE/pCCSL to SHA

Uppaal-smc supports the analysis of stochastic hybrid automata (SHA) that are timed automata whose clock rates can be changed to be constants or expressions depending on other clocks, effectively defining ODEs. This generalizes the model used in previous works [13, 14] where only linear priced automata were handled.

Definition 5 (Semantics of SHA). *The semantics are denoted by a timed Labeled Transition System, which supports the seamlessly transformation from MARTE/pCCSL to SHA. For a SHA H, each location* $(l, v) \in L \times R^X$ *and* $v \models I(l)$*, where the transitions may be of two kinds:*

- *timed transition:* $(l, v) \xrightarrow{d} (l, v')$*, where* $d \in \mathbb{R}_{\geq 0}$ *and* $v' = F(d, v)$
- *event-triggered transition:* $(l, v) \xrightarrow{\alpha} (l', v')$ *, which means the transition is enabled with* $v \models g$ *and* $\varphi \in (v, v')$*.*

For timed transition, the probability density distribution of delay is a uniform distribution or an exponential distribution depending on the invariant of l and $\int \mu_s(t) \cdot dt = 1$ *(μ_s is the probabilistic density function). Let E_l denotes the disjunction of guards g such that* $(l, g, o, -, -) \in E^j$ *for some output o. Let d(l,v) denotes the infimum delay and D(l,v) denotes the supremum delay, i.e.* $d(l,v) = inf\{d \in \mathbb{R}_{\geq 0} : v + R^j \cdot d \models E_l\}$ *and* $D(l,v) = sup\{d \in \mathbb{R}_{\geq 0} : v + R^j \cdot d \models I_l^j\}$*. If* $D(l, v) < \infty$*, the probabilistic density function μ_s means a uniform distribution on* $[d(l, v), D(l, v)]$*. Otherwise, μ_s means an exponential distribution with rate P(l), where I_l^j dosen't put an upper bound on the possible delays out of (l,v) and* $P : L^j \to \mathbb{R}_{\geq 0}$ *is an additional distribution rate. For action transition, the output probability function γ_s over Σ_o^j is the uniform distribution over the set* $\{o : (l, g, o, -, -) \in E^j \wedge v \models g\}$ *whenever this set is non-empty. The detailed semantics of SHA is referred to [13].*

4.2 Transformation from MARTE/pCCSL Models to SHA

Transformation from Mode/State-based MARTE/CCSL behavior to Timed Automata (TA) has been discussed in our previous work [36]. The transformation approach to be presented in this

section will reinforce it by encoding activity behavior, stochastic behavior and continuous behavior into SHA. First, we need to unify all chronometric clocks by converting all physical time units into one. For instance, if we choose the time unit of highest resolution s as the base unit, the conversion has to be conducted through multiplying all clocks in *userProfile* using the unit h by 3600. Then we can encode behaviors (modelled by Activity Diagram or State machine) with pclts according to the semantics of pCCSL. After that, we can map pclts without probabilistic time delay and choice into TA, which addresses two classes of timing constraints: deterministic delay and non-deterministic duration. For instance, the *duration* of *absent* is [8,9](h) that is mapped to an invariant ($t \leq 9$) and a guard ($t \geq 8$). Then, we add probabilistic aspects in transformed TA. For the activity who contains several actions constrained by a clock with exponential distribution, it will be mapped to an location with its exponential parameter λ equalling to the exponential distribution. When we use *GaStep* with the property *prob* to describe the probabilistic choice, it is mapped to two branches with their probabilistic weights. The mapping rules are summarized in Fig. 13. Due to the space limitation of the paper, the detailed correctness demonstration is not given here. But, the correctness of mapping is ensured with the semantics of pCCSL in [36, 40].

Based on the above mapping rules, we can get corresponding SHA models. And then, we can compare and evaluate different refinement versions with UPPAAL-SMC to expose the best alternative one.

4.3 Evaluation with UPPAAL-SMC

With transformed SHA, we conduct evaluation experiment in UPPAAL-SMC. We can get a specific and quantitative analysis using SMC and the evaluation analysis results help us to expose the proper one from different refinement solutions.

To compare and evaluate energy consumption of two refined versions, the following query is used:

$$Pr[energy \leq 1000000](\langle\rangle time \geq 5 * 24)$$

The experiment result is shown in Fig. 14. As we can see, energy consumption of refinement1 is about 50% more than that of refinement2. From the model point of view, the time of staying at the building of refinement1 is $4+4*0.6 = 6.4$hours, which is a little more than that of refinement2. Therefore, the energy consumption of refinement2 should be less than that of refinement1 and our experiment result dose confirm this. So, if we just consider energy-saving, the user profile of refinement version 2 is more suitable.

The details of our experiments can be found in `https://github.com/ECNU-MODANA/AL-Modana.git`.

5 Related Work

This section compares our approach to related works for 1) the modeling of Cyber-Physical Systems in general, 2) the use of MARTE to conduct various kinds of analysis, 3) the use of Statistical Model Checking along with UML-based models.

The main challenges [25] of designing CPSs resides in combining physical and computational models to deal with both the digital embedded systems, their environment and their interactions in a close-loop fashion. In [26], the author acknowledges the need to extend deterministic paradigms with probabilistic constructs to capture the unknown (and unknowable) behaviors in a more precise manner. PTides [41] is mentioned as a successful example of simulation framework that can deal with these challenges. While PTides gives an ad-hoc solution, we strive to rely on industry-standard models as a front-end. We think that UML has a large-enough acceptance in industry to be a good base, and we use some of its extensions to deal with non-functional properties (with MARTE) and systems engineering properties (Parametrics of SysML). PTides, as an analysis tool, could very well be used as a back-end provided that a semantic-preserving transformation is given.

Combining MARTE and SysML to address issues of CPSs was initially suggested in [32]. MARTE was also experimented on industrial-size case studies [16] and proved [21] first, to be up to the task, second, to require additional ad-hoc extensions depending on the kind of analysis that is targeted. MARTE has also been used standalone on academic examples to address some aspects of CPSs, notably in [29] where an extension of StateCharts is proposed and in [28] where

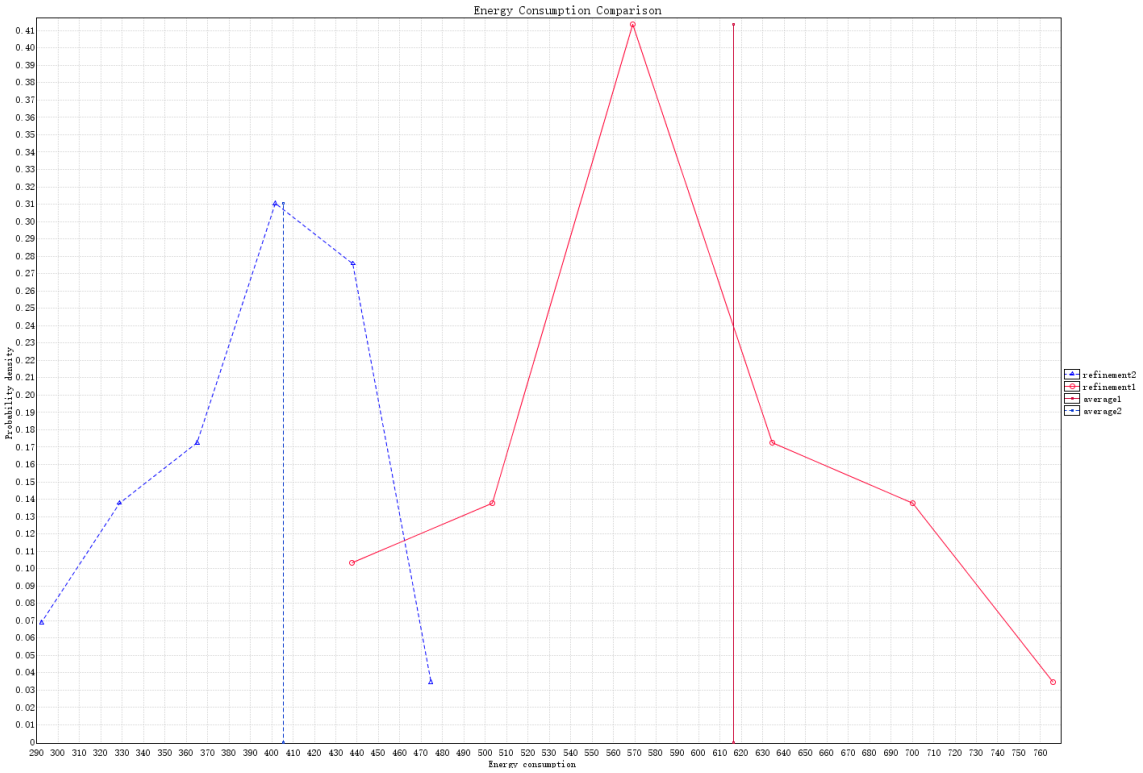


Fig. 14: Energy consumption comparison between two refinement versions

a hybrid version is explored. MARTE is also often used for its ability to describe non-functional properties [4] or as a base to conduct performance analysis [18, 38] or other sort of non-functional analysis like dependability analysis [5]. In literature [34], they use MARTE state machines to model embedded system and transform it to stochastic petri net for energy consumption estimation. In all these examples, only one model of UML is used (like state machine or sequence diagram) while this paper focuses more on finding a practical solution to combine together several UML/SysML/MARTE models. We believe that logical clocks, in a similar way than tagged systems [24], provides the good abstraction level to do that. CCSL [1] then provides a concrete syntax to handle logical clocks. While CCSL is a deterministic declarative polychronous specification language, its pCCSL extension provides new constructs to model the unknown(able) behaviors. Our proposed transformation into stochastic hybrid automata is largely inspired by our previous work to transform CCSL to timed automata [36, 40] however we deal here with new operators of CCSL and with its stochastic extension.

Other works, either based on MARTE [35] or just on UML [9] also use stochastic models to analyze the energy consumption. In this case, the authors rely on Stochastic Petri Nets while in our work we use Stochastic Hybrid Automata. The key difference here again is in the use of probabilistic logical clocks to coordinate the models. However, it is rather a practical matter for us and Stochastic Petri Nets could also be used as a backend instead.

Statistical model checking is a highly scalable simulation-based approach which is useful to bound the probability of making an error by increasing the simulation effort. SMC gets widely accepted in various research areas such as software engineering, in particular for industrial applications [7, 19], or even for solving problems originating from systems biology [12, 22]. Inspired by our previous work [11] which particularly focuses on the analysis of energy consumption of energy-aware building case in different cases to find the most significant factors that influence energy consumption, we attempted to apply MARTE/pCCSL to this case and further evaluate NFP performance of CPSs with SMC.

6 Conclusion

This paper explores an idea of using probabilistic logical clocks in relation to UML models to capture several aspects of CPSs including the stochastic behavior of unknowable events. We believe that using industry-standard is important for a better adoption and we show how to make a consistent use of SysML and MARTE. Clocks through pCCSL are used to coordinate multi-view models. The underlying pCCSL specification is key for the semantic interpretation of models and the transformation into Stochastic Hybrid Automata.

To summarize our contributions 1) MARTE/CCSL has been extended to model stochastic behaviors. The syntax and semantics models of pCCSL are introduced; 2) To show how pCCSL specifications can augment SysML/MARTE models to link together several models and refine the abstract specification; 3) According to the semantics model of pCCSL, we present the transformation process from MARTE/pCCSL to SHA, whose aim is to perform evaluation on the alternative solutions with statistical model checking, which helps designers expose the best solution. The resulting models are analyzed with Uppaal-smc. The process is demonstrated on a simple case study of energy-aware building.

MARTE proposes a subprofile, called Performance Analysis Modeling (PAM), dedicated to performance analysis. We think this contribution is an important step for the integration of MARTE Time model with PAM models. Future work shall consider a more extensive use of PAM, while pCCSL may be kept as much as possible as a lower level intermediate semantic model to ease the transformation from pure UML to other formal models.

7 Acknowledgment

This work is partly supported by NSFC under Grant No. 61472140,61170084, NSF of Shanghai under Grant No. 14ZR1412500, and the Danish National Research Foundation Grant No. 61361136002.

References

1. André, C.: Syntax and semantics of the Clock Constraint Specification Language (CCSL). Research Report 6925, INRIA (May 2009), http://hal.inria.fr/inria-00384077/
2. Arnold, A., Point, G., Griffault, A., Rauzy, A.: The altarica formalism for describing concurrent systems. Fundam. Inform. 40(2-3), 109–124 (1999)
3. Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE 91(1), 64–83 (jan 2003)
4. Berardinelli, L., Bernardi, S., Cortellessa, V., Merseguer, J.: UML profiles for non-functional properties at work: Analyzing reliability, availability and performance. In: 2nd Int. W. on Non-functional System Properties in Domain Specific Modeling Languages (MoDELS/NFPinDSML). CEUR Workshop Proceedings, vol. 553. CEUR-WS.org (2009), http://ceur-ws.org/Vol-553/paper3.pdf
5. Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. Software and System Modeling 10(3), 313–336 (2011), http://dx.doi.org/10.1007/s10270-009-0128-1
6. Bohlender, D., Bruintjes, H., Junges, S., Katelaan, J., Nguyen, V.Y., Noll, T.: A review of statistical model checking pitfalls on real-time stochastic models. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th Int. Symp., ISoLA 2014. Lecture Notes in Computer Science, vol. 8803, pp. 177–192. Springer (2014), http://dx.doi.org/10.1007/978-3-662-45231-8_13
7. Boudjadar, A.J., David, A., Kim, J.H., Larsen, K.G., Mikucionis, M., Nyman, U., Skou, A.: Schedulability and energy efficiency for multi-core hierarchical scheduling systems. ERTS (2014), http://www.erts2014.org/Site/0R4UXE94/Fichier/erts2014_1A1.pdf
8. Boutekkouk, F., Benmohammed, M., Bilavarn, S., Auguin, M.: UML2.0 profiles for embedded systems and systems on a chip (socs). Journal of Object Technology 8(1), 135–157 (2009)
9. Brosig, F., Meier, P., Becker, S., Koziolek, A., Koziolek, H., Kounev, S.: Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. IEEE Trans. Software Eng. 41(2), 157–175 (2015), http://dx.doi.org/10.1109/TSE.2014.2362755
10. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M., Poulsen, D.B., Legay, A., Wang, Z.: Uppaal-smc: Statistical model checking for priced timed automata. arXiv preprint arXiv:1207.1272 (2012)
11. David, A., Du, D., Larsen, K.G., Mikučionis, M., Skou, A.: An evaluation framework for energy aware buildings using statistical model checking. Science China information sciences 55(12), 2694–2707 (2012)

12. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checker for biological systems. International Journal on Software Tools for Technology Transfer (2014), to appear
13. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Vliet, J.V., Wang, Z.: Statistical model checking for networks of priced timed automata. In: FORMATS. pp. 80–96. LNCS, Springer (2011), http://dx.doi.org/10.1007/978-3-642-24310-3_7
14. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Proceedings of the 23rd international conference on Computer aided verification. pp. 349–355. LNCS, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2032305.2032332
15. Deantoni, J., Mallet, F.: Timesquare: Treat your models with logical time. In: Furia, C.A., Nanz, S. (eds.) TOOLS (50). Lecture Notes in Computer Science, vol. 7304, pp. 34–41. Springer (2012)
16. Demathieu, S., Thomas, F., André, C., Gérard, S., Terrier, F.: First experiments using the UML profile for MARTE. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC). pp. 50–57. IEEE (2008), http://dx.doi.org/10.1109/ISORC.2008.36
17. Deshpande, A., Godbole, D.N., Göllü, A., Varaiya, P.: Design and evaluation tools for automated highway systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III: Verification and Control, Proc. of the DIMACS/SYCON Workshop. Lecture Notes in Computer Science, vol. 1066, pp. 138–148. Springer (1995), http://dx.doi.org/10.1007/BFb0020941
18. Espinoza, H., Dubois, H., Gérard, S., Pasaje, J.L.M., Petriu, D.C., Woodside, C.M.: Annotating UML models with non-functional properties for quantitative analysis. In: Bruel, J. (ed.) Satellite Events at the MoDELS 2005 Conference, MoDELS 2005. Lecture Notes in Computer Science, vol. 3844, pp. 79–90. Springer (2005), http://dx.doi.org/10.1007/11663430_9
19. Fang, H., Shi, J., Zhu, H., Guo, J., Larsen, K.G., David, A.: Formal verification and simulation for platform screen doors and collision avoidance in subway control systems. STTT 16(4), 339–361 (2014)
20. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) HSCC. Lecture Notes in Computer Science, vol. 2993, pp. 326–341. Springer (2004)
21. Iqbal, M.Z., Ali, S., Yue, T., Briand, L.: Experiences of applying UML/MARTE on three industrial projects. In: 15th Int. Conf. on Model Driven Engineering Languages and Systems. pp. 642–658. MODELS'12, Springer-Verlag, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33666-9_41
22. Jiang, Z., Pajic, M., Moarref, S., Alur, R., Mangharam, R.: Modeling and verification of a dual chamber implantable pacemaker. In: Flanagan, C., König, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 18th Int. Conf., TACAS 2012. Lecture Notes in Computer Science, vol. 7214, pp. 188–203. Springer (2012), http://dx.doi.org/10.1007/978-3-642-28756-5_14
23. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
24. Lee, E.A., Sangiovanni-Vincentelli, A.L.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(12), 1217–1229 (December 1998)
25. Lee, E.A.: Cyber physical systems: Design challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008). pp. 363–369. IEEE Computer Society (May 2008), http://dx.doi.org/10.1109/ISORC.2008.25
26. Lee, E.A.: The past, present and future of cyber-physical systems: A focus on models. Sensors 15(3), 4837–4869 (2015), http://www.mdpi.com/1424-8220/15/3/4837
27. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Runtime Verification. pp. 122–135. Springer (2010)
28. Liu, J., Liu, Z., He, J., Mallet, F., Ding, Z.: Hybrid MARTE statecharts. Frontiers of Computer Science 7(1), 95–108 (2013)
29. Liu, Z., Liu, J., He, J., Ding, Z.: Spatio-temporal UML statechart for cyber-physical systems. In: 17th IEEE Int. Conf. on Engineering of Complex Computer Systems, ICECCS. pp. 137–146. IEEE Computer Society (2012), http://doi.ieeecomputersociety.org/10.1109/ICECCS.2012.36
30. Mallet, F., de Simone, R.: Correctness issues on MARTE/CCSL constraints. Science of Computer Programming (2015), http://www.sciencedirect.com/science/article/pii/S0167642315000519
31. Ptolemaeus, C.: System Design, Modeling, and Simulation: Using Ptolemy II. Ptolemy.org (2014)
32. Selic, B., Gerard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE. Elsevier (2013)
33. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D. (eds.) Computer Aided Verification, 16th Int. Conf., CAV. Lecture Notes in Computer Science, vol. 3114, pp. 202–215. Springer (2004), http://dx.doi.org/10.1007/978-3-540-27813-9_16
34. Shorin, D., Zimmermann, A., Maciel, P.: Transforming uml state machines into stochastic petri nets for energy consumption estimation of embedded systems. In: Sustainable Internet and ICT for Sustainability (SustainIT), 2012. pp. 1–6 (Oct 2012)

35. Shorin, D., Zimmermann, A.: Formal description of an approach for power consumption estimation of embedded systems. In: 24th Int. W. on Power and Timing Modeling, Optimization and Simulation, PATMOS). pp. 1–10 (2014), http://dx.doi.org/10.1109/PATMOS.2014.6951890
36. Suryadevara, J., Seceleanu, C., Mallet, F., Pettersson, P.: Verifying MARTE/CCSL mode behaviors using UPPAAL. In: Software Engineering and Formal Methods, pp. 1–15. Springer (2013)
37. Tomlin, C., Pappas, G.J., Lygeros, J., Godbole, D.N., Sastry, S.: Hybrid control models of next generation air traffic management. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S. (eds.) Hybrid Systems IV. Lecture Notes in Computer Science, vol. 1273, pp. 378–404. Springer (1996), http://dx.doi.org/10.1007/BFb0031570
38. Tribastone, M., Gilmore, S.: Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In: Avritzer, A., Weyuker, E.J., Woodside, C.M. (eds.) 7th Int. W. on Software and Performance, WOSP 2008. pp. 67–78. ACM (2008), http://doi.acm.org/10.1145/1383559.1383569
39. Younes, H.L.S., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. 204(9), 1368–1409 (2006), http://dx.doi.org/10.1016/j.ic.2006.05.002
40. Zhang, Y., Mallet, F., Chen, Y.: Timed automata semantics of spatial-temporal consistency language STeC. In: Theoretical Aspects of Software Engineering Conference, TASE. pp. 201–208. IEEE (2014), http://dx.doi.org/10.1109/TASE.2014.10
41. Zou, J., Matic, S., Lee, E.A., Feng, T.H., Derler, P.: Execution strategies for PTIDES, a programming model for distributed embedded systems. In: 15th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS. pp. 77–86. IEEE Computer Society (2009), http://dx.doi.org/10.1109/RTAS.2009.39